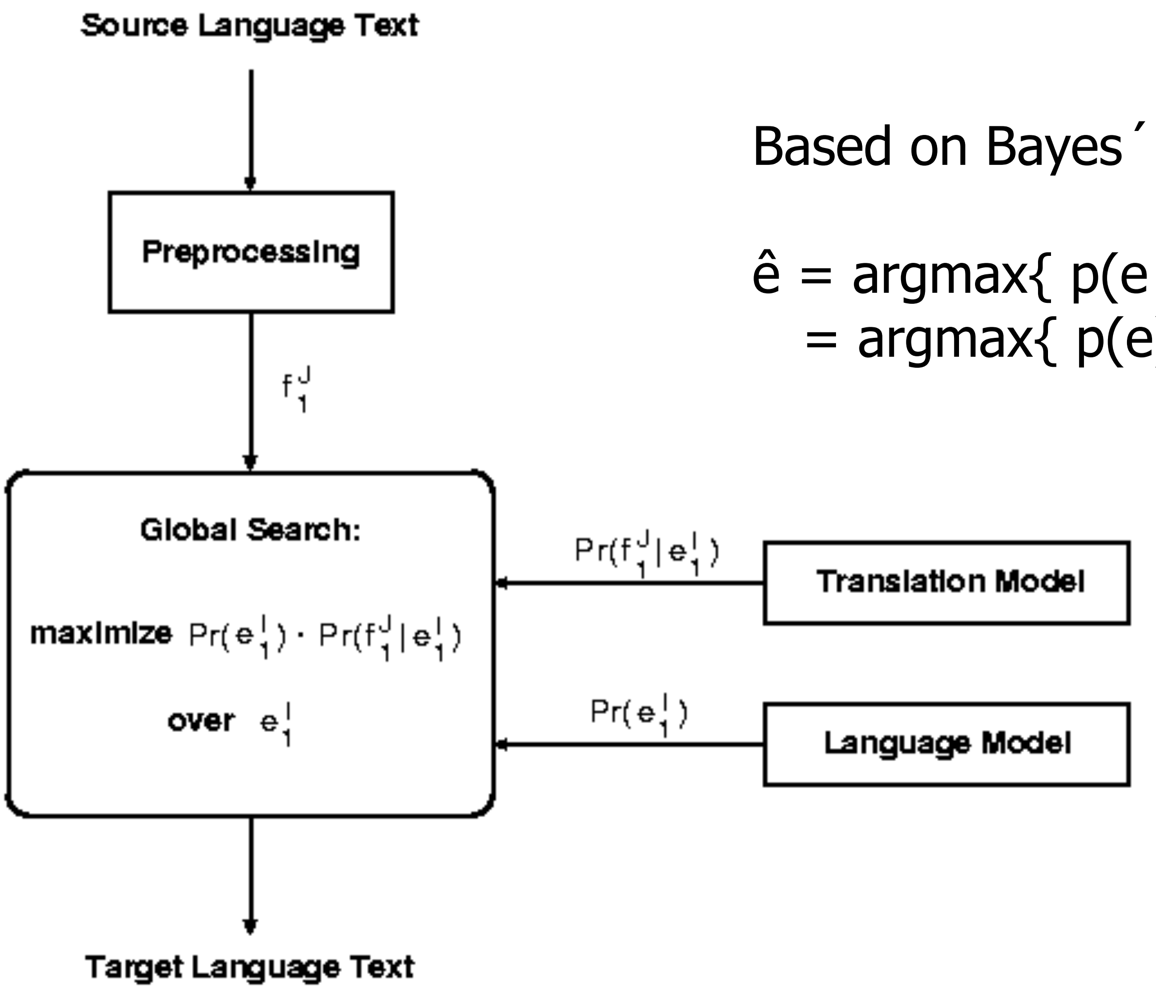
Language Modeling for Machine Translation

Overview

- Introduction
- N-Gram Models
- Measuring Preformance
- Count Smoothing
- Interpolation / Backoff
- Additional LM Types
- Special Problems

SMT Architecture



Based on Bayes´ Decision Rule:

$$\hat{e} = \text{argmax}\{\ p(e \mid f)\ \}$$
$$= \text{argmax}\{\ p(e)\ p(f \mid e)\ \}$$

Language Model

- Main goal: Produce fluent English
  - Not only correct meaning
  - Some notion of grammaticality

- Also guidance for other problems
  - Word order
    - p(the house is small) = high
    - p(small the house is) = low
  - Word translation:
    - P(I am going home) = high
    - p(I am going house) = low

Language Model

- Formally:
  - Function English Sentence -> Probability that Sentence was produced by English Speaker
- indicates whether a sentence is **good English** (or German, …):

- Useful for many *natural language processing* applications, such as machine translation, speech recognition, part-of-speech tagging, parsing and information retrieval
- Approaches:
  - deterministic, e.g. finite state grammars
  - A **statistical language model** assigns a probability to a sequence of words $P(w_{1..n})$ by means of a probability distribution

Statistical Language Model

- Compute probability
$$p(W) = p(w_1, w_2, \ldots, w_N)$$
  - Use statistics from large corpora

- Simple approach: look at a large text database (1*10^9 Sentences).
  - Count(“how's it going?”) = 76,413
  - p(how's it going?) = 76,413/1,000,000,000 = 0.000076413
- Problem: sparse data
  - many perfectly good sentences will be assigned a **p(e)** of zero, because they have never been seen before

Statistical Language Model

- Advantage:
  - Trainable on Large Text Databases
  - Prediction ‘Soft’ (Probabilities)
  - Can be combined with translation model
- Problem:
  - Need Large Text Database for each Domain

Statistical Language Model

- Beak up into prediction of one word
- Decompose probability

$$p(w_1, w_2, \ldots, w_N) = p(w_1) * p(w_2 \mid w_1) * \ldots * p(w_N \mid w_1, w_2, \ldots, w_{N-1})$$

- Product of word probabilities given history
- Example:
  - Das tut mir aber…
    - …Leid, Weh
    - .. Haus (?)

Data Sparsity

- Estimating $p(w_N \mid w_1, w_2, \ldots, w_{N-1})$
- Example:
  - 64,000 words
  - average sentence lengths of 25 words
  - number of possible histories ($64{,}000^{25} > 10^{120}$)
- Cannot pre-compute every history
- Solutions:
  - compute *P*(*w* | history) "on the fly" (rarely used, very expensive)
    - Many will not occur -> how to estimate probability
  - replace the history by one out of a limited feasible number of classes

$$p(w_N \mid w_1, w_2, \ldots, w_{N-1}) = p(w_N \mid C(w_1, w_2, \ldots, w_{N-1}))$$

Classification of Word Sequence Histories

- grammatical content (phrases like noun-phrase, etc.)
- POS = part of speech of previous word(s)
- semantic meaning of previous word(s)
- context similarity (word(sequence)s that are often observed in similar contexts are treated equally, e.g. weekdays, people's names etc.)
- apply some kind of automatic clustering (agglomerative or divisive)
- Markov assumption:
  - Word probability depend only on n preceding words
  - Wrong, but good approximation

$$p(w_N \mid w_1, w_2, \ldots, w_{N-1}) = p(w_N \mid w_{N-k+1}, \ldots, w_{N-1})$$

N-Gram

- Example: the house is small

- Unigram: P(is)
  - No history
  - prior probabilities of word observation
- Bi-gram: P(is| house)
- Tri-gram: P(is | the house)
- 4-gram: P(is | <s> the house)

Trigram Language Model

- p(I like snakes that are not poisonous) =

  p(I | <s> <s>) *

  p(like | <s> I) *

  p(snakes | I like) *

  p(that | like snakes) *

  p(are | snakes that) *

  p(not | that are) *

  p(poisonous | are not) *

  p(</s> | not poisonous) *

  p(</s> | poisonous </s>)

Estimate N-gram Probabilities

- Maximum Likelihood estimation

$$p(w_3 \mid w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_{w} count(w_1, w_2, w)}$$

Estimate N-gram Probabilities

- Example:
  - Count from EPPS corpus

the green (total: 1748)

word	c.	prob.
paper	801	0.458
group	640	0.367
light	110	0.063
party	27	0.015
ecu	21	0.012

the red (total: 225)

word	c.	prob.
cross	123	0.547
tape	31	0.138
army	9	0.040
card	7	0.031
,	5	0.022

the blue (total: 54)

word	c.	prob.
box	16	0.296
.	6	0.111
flag	6	0.111
,	3	0.056
angel	3	0.056

Are N-Grams a good solution

Are Bigrams / Trigrams any good?

First experiment:

- 1.5 million words used for training
- 300,000 words used for testing
- restricted to 1,000 most frequent words

=> 23% of trigrams occurring in test corpus were absent from training corpus

Second experiment (bag of words):

- take any meaningful 10-word sentence (from dictation task)
- scramble the words into an arbitrary order
- find most probable order with trigram model

=> 63% perfect word-by-word reconstruction
79% reconstruction that preserves meaning

The Bag of Words Experiment

Most likely trigram sequences from randomly scambled dictated sentence:

- I expect that the output will improve with experience.
  I expect that the output will improve with experience.

- would I report directly to you?.
  I would report directly to you?.

- now let me mention some of the disadvantages.
  let me mention some of the disadvantages now.

- these people have a fairly large rate of turnover.
  of these people have a fairly large turnover rate.

- exactly how this might be done is not clear.
  clear is not exactly how this might be done.

A Word Guessing Game

What do we learn from the word guessing game?

- for some histories the number of expected words is rather small.

- for some histories we can make virtually no prediction about the next word.

- the more words fit at some point the more difficult it is to recognize the correct one
(more errors are possible)

- the difficulty of recognizing a word sequence is correlated with the "branching degree"

Measuring the Quality of Language Models

- Many design decisions:
  - How much and which training data?
  - N-Gram Length
  - Smoothing Technique

- Need to measure quality

Measuring the Quality of Language Models

- How to test if LM1 or LM2 is better?
- Obvious approach:
  - Use both in MT system
  - Use LM which leads to better MT performance
- Problem:
  - Performance depend on TM and interaction between TM and LM
  - Time-consuming
- Independent Measure

Measuring the Quality of Language Models

- Assumption:
  - Good English -> high Probability
  - Bad English -> low Probability
- Measure:
  - Held out data (assumed to be good English)
  - Calculate Probability of this data
  - Higher Probability -> Better language model
- Use Perplexity of test data

Measuring the Quality of Language Models

- Cross Entropy:

$$H(p_{LM}) = -\frac{1}{n}\log(p_{LM}(w_1, w_2, \ldots w_N))$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\log(p_{LM}(w_i \mid w_1, ..w_{i-1}))$$

- Perplexity:

$$PP = 2^{H(p_{LM})}$$

  - interpret it as the "branching factor" of the language

Example

prediction	p_{LM}	$-\log_2 p_{\text{LM}}$
$p_{\text{LM}}(i \mid </s><s>)$	0.109	3.197
$p_{\text{LM}}(would \mid <s> i)$	0.144	2.791
$p_{\text{LM}}(like \mid i\ would)$	0.489	1.031
$p_{\text{LM}}(to \mid would\ like)$	0.905	0.144
$p_{\text{LM}}(commend \mid like\ to)$	0.002	8.794
$p_{\text{LM}}(the \mid to\ commend)$	0.472	1.084
$p_{\text{LM}}(rapporteur \mid commend\ the)$	0.147	2.763
$p_{\text{LM}}(on \mid the\ rapporteur)$	0.056	4.150
$p_{\text{LM}}(his \mid rapporteur\ on)$	0.194	2.367
$p_{\text{LM}}(work \mid on\ his)$	0.089	3.498
$p_{\text{LM}}(. \mid his\ work)$	0.290	1.785
$p_{\text{LM}}(</s> \mid work\ .)$	0.99999	0.000014
	average	2.634

Count Smoothing

- Problem: N-Gram has not been seen in training
  - Maximum likelihood estimation -> Probability is 0
  - Quite harsh
  - Not very useful
    - OOV in sentence -> All Hypothesis have probability of 0
- Assign also positive probabilities to unseen n-grams
  - Higher order n-grams -> even more important
- Empirical counts:
  - Counts seen in the training data
- Expected counts:
  - Counts in previously unseen text

Add – One Smoothing

- Simpelst approach:
  - Add fixed number(1) to every count
  - No counts are zero
  - No zero probabilities

$$p = \frac{c}{n} \rightarrow p = \frac{c+1}{n+v}$$

    - V = total number of possible n-grams
  - Problem:
    - Example:
      - Voc-Size = 86,700
      - Possible bi-grams: 86,700^^2 = 7,516,890,000 (7.5 billion)
      - Corpus Size: 30 million
    - To much weight to unseen examples

Add – α Smoothing

- Add α < 1 instead:

$$p = \frac{c}{n} \rightarrow p = \frac{c+\alpha}{n+\alpha v}$$

- How to find α:
  - Optimize on Preplexity
  - Match between adjusted counts and test counts

Add – α Smoothing - Example

Count	Adjusted count		Test count
c	$(c+1)\frac{n}{n+v^2}$	$(c+\alpha)\frac{n}{n+\alpha v^2}$	t_c
0	0.00378	0.00016	0.00016
1	0.00755	0.95725	0.46235
2	0.01133	1.91433	1.39946
3	0.01511	2.87141	2.34307
4	0.01888	3.82850	3.35202
5	0.02266	4.78558	4.35234
6	0.02644	5.74266	5.33762
8	0.03399	7.65683	7.15074
10	0.04155	9.57100	9.11927
20	0.07931	19.14183	18.95948

Deleted Esitmation

- Estimate True Counts using held-out data
  - Split data in two parts
  - Counts in training of n-gram c=r
  - Number of N-grams with training count r: N_r
  - Total number of n-grams with training count r in test: T_r

$$r^* = \frac{T_r}{N_r}$$

  - Use r* instead of original counts in probability estimation
- Switch test and training data
  - Use average

$$r^* = \frac{(T_{r1} + T_{r2})}{(N_{r1} + N_{r2})}$$

Deleted Esitmation- Example

Count	Count of count	Count in held-out	Exp. count
r	N_r	T_r	$E[r] = \frac{T_r}{N_r}$
0	7,515,623,434	938,504	0.00012
1	753,777	353,383	0.46900
2	170,913	239,736	1.40322
3	78,614	189,686	2.41381
4	46,769	157,485	3.36860
5	31,413	134,653	4.28820
6	22,520	122,079	5.42301
8	13,586	99,668	7.33892
10	9,106	85,666	9.41129
20	2,797	53,262	19.04992

- r* = 353.383/753.777=0.469

Good – Turing Smoothing

- Idea: What does it mean if an n-gram occurs c times
- Mathematical analysis:
  - Calculate expected count r^* using count of counts

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

- Advantages:
  - Quite simple to calculate
- Disadvantage:
  - Unreliable for large r
    - Curve - fitting
    - Only use for small r

Good – Turing Smoothing - Example

Count	Count of counts	Adjusted count	Test count
r	N_r	r^*	t
0	7,514,941,065	0.00015	0.00016
1	1,132,844	0.46539	0.46235
2	263,611	1.40679	1.39946
3	123,615	2.38767	2.34307
4	73,788	3.33753	3.35202
5	49,254	4.36967	4.35234
6	35,869	5.32928	5.33762
8	21,693	7.43798	7.15074
10	14,880	9.31304	9.11927
20	4,546	19.54487	18.95948

Evaluation

Smoothing method	Perplexity
Add-one	383.2
Add-α ($\alpha = 0.00017$)	113.2*
Deleted estimation	113.4
Good-Turing	112.9

Interpolation and Backoff

- General:
  - Longer context -> Better language models
- Problem:
  - Limited data -> More n-grams not observed
- Absolute Discounting:
  - Treats all n-grams equally

- Example:
  - Scottish beer drinkers
  - Scottish beer eaters
- Both have not been seen
  - Treated equally

Interpolation and Backoff

- Backoff to bi-grams:
  - Beer drinkers
  - Beer eater

- First may have been see
  - Can use these counts

Interpolation

- Make use of more robust estimated lower n-grams

$$\begin{aligned} p_I(w_3 \mid w_1 w_2) &= \lambda_1 p_1(w_3) \\ &+ \lambda_2 p_2(w_3 \mid w_2) \\ &+ \lambda_3 p_3(w_3 \mid w_1 w_2) \end{aligned}$$

$$\forall \lambda_n : 0 \leq \lambda_n \leq 1$$

$$\sum_n \lambda_n = 1$$

- Optimize weights on held-out data set

Recursive Interpolation

- Weight depend on history

$$p_n^I(w_i|w_{i-n+1},...,w_{i-1}) =$$
$$= \lambda_{w_{i-n+1},...,w_{i-1}} \; p_n(w_i|w_{i-n+1},...,w_{i-1}) +$$
$$+ (1 - \lambda_{w_{i-n+1},...,w_{i-1}}) \; p_{n-1}^I(w_i|w_{i-n+2},...,w_{i-1})$$

- Group histories:
  - According to frequency

Backoff

- Trust the highest n-gram with counts

$$p_n^{BO}(w_i|w_{i-n+1},\ldots,w_{i-1}) =$$

$$= \begin{cases} d_n(w_{i-n+1},\ldots,w_{i-1})\, p_n(w_i|w_{i-n+1},\ldots,w_{i-1}) \\ \qquad\qquad \text{if count}_n(w_{i-n+1},\ldots,w_i) > 0 \\ \\ \alpha_n(w_{i-n+1},\ldots,w_{i-1}))\, p_{n-1}^{BO}(w_i|w_{i-n+2},\ldots,w_{i-1}) \\ \qquad\qquad \text{else} \end{cases}$$

- Adjust weights and discounting

Backoff with Good-Turing Smoothing

- In Good-Turing Smoothing:

$$count^*(w_1 w_2) < count(w_1 w_2)$$

- Use for weights:

$$d(w_2 \mid w_1) = \frac{count^*(w_1 w_2)}{count(w_1 w_2)}$$

- Zero count n-grams stay the same
- Discounting:

$$a(w_1) = 1 - \sum d(w_2 \mid w_1)$$

Diversity of Predicted Words

- Example: spite and constant
  - Both occur 993 times in Europal corpus
    - 9 words follow spite (mostly spite of)
    - 415 different word follow constant

- More likely to see new bigram starting with constant than spite

- Witten-Bell smoothing

Witten-Bell Smoothing

- Recursive interpolation method
- Number of possible extensions of history:

$$N_{1+}(w_1,\ldots,w_{n-1},*) = |\{w_n : c(w_1,\ldots,w_{n-1},w_n) > 0|$$

- Lambda parameters:

$$1-\lambda_{w_1,\ldots,w_{n-1}} = \frac{N_{1+}(w_1,\ldots,w_{n-1},*)}{N_{1+}(w_1,\ldots,w_{n-1},*) + \sum_{w_n} c(w_1,\ldots,w_{n-1},w_n)}$$

Witten – Bell - Example

$$1 - \lambda_{spite} = \frac{N_{1+}(\text{spite}, \bullet)}{N_{1+}(\text{spite}, \bullet) + \sum_{w_n} c(\text{spite}, w_n)}$$

$$= \frac{9}{9 + 993} = 0.00898$$

$$1 - \lambda_{constant} = \frac{N_{1+}(\text{constant}, \bullet)}{N_{1+}(\text{constant}, \bullet) + \sum_{w_n} c(\text{constant}, w_n)}$$

$$= \frac{415}{415 + 993} = 0.29474$$

Diversity of Histories

- Example: York
  - Quite frequent in Europarl(477 times, like foods, indicates, …)
  - -> high uni-gram probability
- Nearly always follows New (473)

- Usage of Uni-gram model:
  - Only used if bigram not there

- -> York unlikely to be second word of unseen n-gram
- Back-off unigram -> lower probability for York

Kneser-Ney Smoothing

- Use diversity of histories for lower order n-grams
- Count of history for a word:

$$N_{1+}(*w) = |\{w_i : c(w_i, w) > 0|$$

- Maximum Likelihood estimation

$$p_{ML}(w) = \frac{c(w)}{\sum_i c(w_i)}$$

- Knesery Ney smoothing

$$p_{KN}(w) = \frac{N_{1+}(*w)}{\sum_i N_{1+}(*w_i)}$$

Modified Kneser-Ney Smoothing

- Most commonly use smoothing technique
- Different stategies for highest order and other n-grams:
- Highest order n-grams:
  - Absolute discounting with three different values (1,2,3+)
  - Similar to Witten-Bell for backoff
- Lower order n-grams:
  - Discounting of the histories using again 3 absolute discounting values
- Backoff and Interpolated version

Evaluation

Smoothing method	bigram	trigram	4-gram
Good-Turing	96.2	62.9	59.9
Witten-Bell	97.1	63.8	60.4
Modified Kneser-Ney	95.4	61.6	58.6
Interpolated Modified Kneser-Ney	94.5	59.3	54.0

Different Kinds of Language Models

So far, we have analyzed static *n*-grams. There also are:

- cache language models (constantly adapting to a floating text)
- trigger language models (can handle long distance effects)
- POS-based language models, LM over POS tags
- class-based language models based on semantic classes
- multilevel *n*-gram language models (mix many LM together)
- interleaved language models (different LM for different parts of text)
- morpheme-based language models (separate words into core and modifyers)
- context free grammar language models (use simple and efficient LM-definition)
- decision tree language models (handle long distance effects, use rules)
- HMM language models (stochastic decision for combination of independent LMs)

Cache Language Models

Observation: When using a machine translation system (e.g. for news texts) the topics can change at arbitrary points.

- **Idea:**
  Use a static and a dynamic component of the language model.
  Constantly update the dynamic component.
- **Static Component:**
  $P_S(w_k \mid w_{k-(n-1)} \ldots w_{k-1})$ i.e. the usual trained language model.
- **Dynamic Component:**
  complete *n*-gram language model constructed from the text translate so far, e.g.:
  $P_D(w_k \mid w_{k-(n-1)} \ldots w_{k-1}) = 0.5 \cdot f(w_k) + 0.25 \cdot f(w_k \mid w_{k-1}) + 0.25 \cdot f(w_k \mid w_{k-2}\, w_{k-1})$
- **Total Language Model:**
  $P_T = \lambda \cdot P_D + (1- \lambda) \cdot P_S$
- **Variation:** Compute P_D on window of last *l* words.

Trigger Language Models

Karlsruher Institut für Technologie

Observation: Often, the probability of a word depends on some word far in the history. Often, when a content-carrying word occurs in a text it is likely to occur again some time later.

- **Idea:**
  Use a standard interpolated *n*-gram language model.
  But constantly update the unigrams $P(w_k)$ for some words w_k.

- **Trigger**
  For each word w define a trigger list $L(w)$ (possibly weighted) of words that are likely to occur some time later.

  E.g. L(MONEY) = {BANK, SAVINGS, ACCOUNT, DOLLARS, COST}.
  Then, for each word v in $L(w)$ increase $P(v)$ by some value.

Multilevel Language Models

- **Idea:**
  Use a language model for word sequences LM_W,
  one for phrases LM_P,
  and one for sentences LM_S
  trained as regular *n*-gram grammars over their corresponding segments of speech.

- **Combination**
  Let the total language model be a linear combination
  of the three independent language models:
  $LM_T = \lambda \cdot LM_W + \lambda \cdot LM_P + \lambda \cdot LM_S$

Morpheme-Based Language Models

- **Observation:**
  Often, in inflecting languages, the probability for a word can be estimated more robustly for the core of the word than for the word together with its pre- and suffixes.

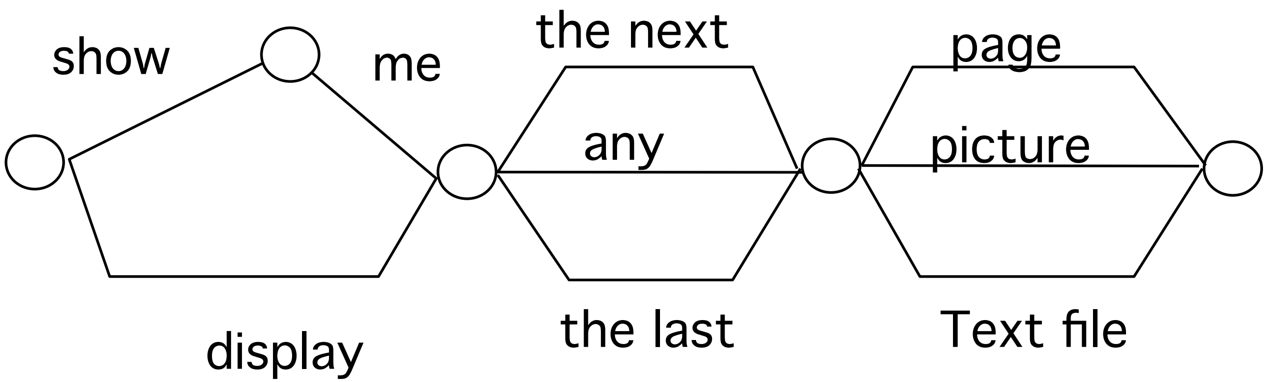
  Often, the gender- tempus- numerus- and case-identifying suffixes depend on the corresponding form of the preceeding adjective or article.

- **Idea:**
  Don't compute language model on sequences of words but on sequences of word-building fragments (morphemes):

$$P(w_1\ w_2\ \ldots) = P(w_{11}\ w_{12}\ w_{13}\ w_{21}\ w_{22}\ \ldots)$$

Language Models: Grammar Based

- Write Grammar of Possible Sentence Patterns



- Advantages:
  - Long History / Context
  - Don’t Need Large Text Database (Rapid Prototyping)
- Problem:
  - Work to Write Grammars
  - Rigid: Only Programmed Patterns can be Recognized

Context-Free Grammar Language Models

Why use context-free grammars (CFG) instead of *n*-grams?

- The rules of the grammar can be written by expert without the need for lots of training text data.
- CFGs are powerful enough to represent large parts of natural languages.
- CFGs are constrained enough to allow efficient search space reduction.
- If interpreted as finite state automaton, then the state transition sequence can also be used for efficient parsing (semantic analysis) of the sentence.

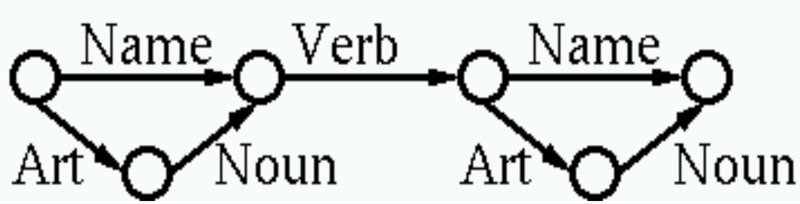
S := NP VP

VP := Verb NP

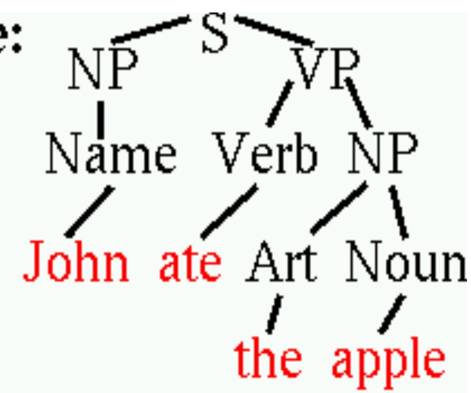
NP := Name

NP := Art Noun

Transition Network:



Parse Tree:



Special Problems with Spontaneous Speech

In spontaneous speech, we often observe words that blow up the word sequence but in most cases don't influence the language model:

- silence between words can be optionally inserted or omitted
- emphatic pauses: UHs, UMs, ...
- filler words and phrases: YEAH, YOU KNOW, ...
- aborted or stuttered words
- non speech sounds: breathing, lip smacks, tong clicks, ...

Such words are narrowing the context of *n*-grams, sometimes even push out the entire content-carrying context.

Approaches:

- ignore any problem, treat spontaneous effects like regular words
- increase context width (use four-grams, etc.)
- skip spontaneous effects in the history of $P(w \mid \text{history})$

Special Problems with Unknown Words

Question: How do we incorporate unknown words into the language model?

Approaches:

- Rewrite all words in the training text that occur only once with the word "UNK",
  treat every unknown word in the recognition run as if it was "UNK".
- Use a class-based language model, assign a class to every unknown word and use the class probabilities.
- Optionally add new word into vocabulary and use cache language model to improve the parameters of frequently occurring new words.

Special Problems with Different Languages

- Languages differ in their degree of inflection.
- Highly inflecting languages pose more problems to language modeling.

(Korean allows tens of thousands of forms of the same verb)
=> use morpheme-based language models, and syntactic/ semantic classes

- Languages differ in their potential to form new words.
- E.g. German allows arbitrary compounding of nouns.

=> decompose compound nouns for calculating the language model

- Some languages have a different or not exactly specified notion of words.
- (Given 曲阜孔子博物院

Where does a word start and where does it end?)
=> use syllable based language models